Database Management System Policies and Requirements

This module is the second of four modules that describe the use of the Trusted Database Interpretation (TDI) of the Trusted Computer System Evaluation Criteria (TCSEC) for database product evaluation and certification. The basic terms and concepts presented in the TDI are summarized in TDI Module One. This module describes security policies and database requirements that can be supported by a trusted database management system (DBMS). TDI Module Three describes various architectural approaches for building a trusted DBMS. TDI Module Four describes other database security issues that are not covered by the TDI but are important issues in database security, such as inference, aggregation, and database integrity.

Module Learning Objectives

This module describes how an MLS DBMS can enforce Mandatory and Discretionary Access Control on the objects that it protects, as well as the supporting policies of Identification and Authentication (I&A) and Audit. The student will be introduced to the types of objects commonly encountered in trusted databases as well as the types of operations subject to DAC controls and the MAC requirements for specific database operations. Upon completion of this module the student should:

1) understand the issues surrounding MAC policies and their enforcement in a trusted DBMS.

2) understand what DAC policies are normally enforced and what policy areas are left up to the DBMS developer.

3) understand how I&A and auditing must be done in a trusted DBMS to meet the TCSEC requirements.

4) be familiar with the issues surrounding the enforcement of both secrecy and integrity policies in a trusted DBMS.

5) be familiar with the security issues surrounding the need for the standard database capabilities of concurrency and recovery.

Overview

There are a wide variety of security policies that can be enforced by a trusted DBMS. This module concentrates on security policies that are required by the TCSEC (i.e., MAC, DAC) and integrity policies, since they are so critical to databases. It also covers supporting policies such as I&A and Audit that are levied on DBMSs to support these security policies.

The fundamental concepts of MAC and DAC for databases are similar to that for operating systems (OS)s. The differences lie in the granularity and types of objects that must be protected and the relationships that exist between these objects. In OSs, objects generally are not tightly coupled with other objects,

with the possible exception of directories and the files they contain. In contrast, most database objects are related to and nested within other objects in the database. These relationships can make enforcing effective access control policies a challenge.

Before beginning, some terminology needs to be defined as a basis for discussion. These terms are from the relational model [JCOD70]. There are several other database models that could be used to design a DBMS. However, because of the prevailing use of this model in industry, this terminology will be used here. The ANSI Standard database language SQL is based on the relational model.

In the relational model, a database is viewed as a collection of information. The structure of the database is described by a schema. The schema contains the definitions of the tables and views of the database. The actual data of the database is stored in tables. These tables are described by table definitions which are part of the database schema. A table definition defines the number of attributes that are present in the table, the name of each of these attributes, and what type of information can be stored in each attribute. The actual information stored within a table is organized into tuples. A tuple is composed of a set of elements, one element per attribute. Tuples are sometimes called database rows in the literature. The tables, tuples, and elements that are stored in the database make up what are called the base relations in the database. They are what is actually stored in the database. These terms are depicted in Figure 2-1. The relational model also supports

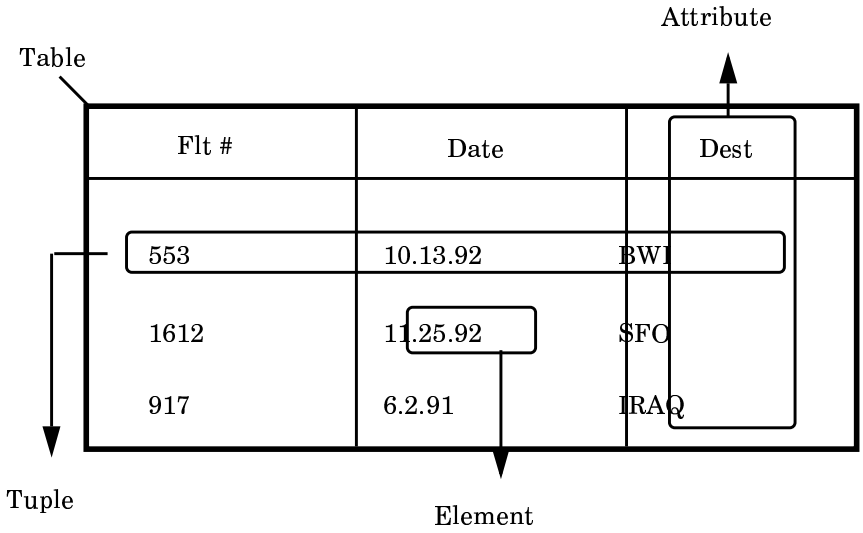

Figure 2-1: Relational Model Terminology

the ability to create views (named queries) of the data in the base relations. Views are defined by view definitions, which are part of the database schema. A view is a way of looking at the information stored in the database in a different way from how it is stored in the base relations. Views do not change

the underlying structure of the base relations. When access through a view is requested, queries (as defined by its definition) are sent to the DBMS to retrieve the information that can be seen through the view. Changes made to data seen through a view alter the underlying base relations.[1]

MAC in a Trusted DBMS

A database contains table definitions, view definitions, and tables. Table and view definitions are typically stored in the database in tables owned by the DBMS. Database objects must also be protected by the security policy enforced by the DBMS, which must include MAC at B1 and above. The granularity of this protection is left up to the DBMS designer. This policy must take into account the relationships that exist between these objects and how they are stored in the database. These topics are discussed in the following sections:

MAC Granularity

The MAC policy identifies atomic objects to which access will be controlled. The granularity of MAC is based on the smallest distinguishable component of the database which is labeled. This can be the database, table, tuple, attribute, or element level. Typically, tuple or element level labeling is done in order to provide sufficient granularity. If the database is chosen as the atomic level, then everything within it would be a single level. This degenerate case is equivalent to a single level database stored in files on a trusted operating system. Choosing tables as the atomic level provides the ability to differentiate the data to some degree, but in general, is an insufficient level of granularity to be useful. Element level MAC controls provide the greatest flexibility. However, because of their complexity and the processing and storage overhead required to support element level labeling, tuple level labeling is frequently provided instead.

Regardless of the level of granularity selected, all objects must be labeled either explicitly or implicitly. If the granularity selected is above the element level (e.g., table, attribute, tuple), then all objects contained within the labeled object (e.g., tuple, element) and any objects they contain are implicitly labeled with the label of the containing object. For example, if the granularity chosen is at the table level, then all tuples contained in the table, and all elements contained in those tuples are implicitly marked with the label of the table. Objects that contain labeled objects may be labeled to indicate the minimum level required to access the container. Names and definitions of tables, views and entire databases must also be considered objects and labeled accordingly. The following discusses the labeling options for each object type:

---

1. The ability to alter data through a view is constrained based on the structure of the view and the underlying table. These constraints are described in [TDAT89].

Elements

Elements are labeled if MAC is provided at the granularity of the element. Otherwise they are implicitly labeled with the label of the containing tuple, attribute or table depending on the granularity of MAC supported.

Attributes

Attributes define columns of a table. A labeling scheme could choose to label the columns of the table by assigning a label to each attribute, thereby labeling all elements within that column. However, this scheme is generally not used because it does not provide the type of labeling that is used to model real world relationships. Because of this, attribute level labeling is not discussed further within this module.

Tuples

Tuples are labeled if MAC is provided at the granularity of the tuple. If elements are labeled then tuples do not need explicit labels because their sensitivity can be derived from the sensitivities of the elements they contain. Since a tuple and all of its elements are created simultaneously (i.e., all elements must exist even though they may have null values) then the implied label can always be computed. If MAC is done at the granularity of the table, then the tuple is implicitly labeled with the label of the table it is contained in.

Tables

If MAC is done at the granularity of tables, each table is labeled and all tuples and elements contained in the table are implicitly labeled at that level. However, MAC granularity is normally provided at the tuple or element level.

Table Definitions

Table definitions must be labeled if they can be created at more than one level. These labels also provide a minimum level needed to access the table. The meaning of a table definition label is discussed more in the section entitled Database and Table Labels.

Database

When a database is created it is basically an empty container with a name. The name must be labeled as required by the MAC policy. This label defines the minimum sensitivity level required to access the database. This is discussed in more detail in the following section.

Database and Table Labels

The following hypothetical implementation illustrates the typical meanings of and relationships between the sensitivities of databases and the tables they contain. Other database implementations are possible.

When a database is created it must be labeled. Since a database is initially empty, the only thing the label refers to is the metadata defining the database. To be able to read/open the database, a user's current session level must dominate this database label. Whenever a new table is created, the table definition must be marked at the user's current session level because the session level may be more sensitive than the sensitivity label of the database name. In the operating system realm, any modifications to a file by a more sensitive process would require that the file be upgraded to the sensitivity of the user doing the modification. However, since the granularity of MAC for a trusted DBMS is finer than that in a OS, a label can be placed on what was modified (i.e., the new table definition) rather than upgrade the entire container (the database). One can see that the label on the database indicates the minimum sensitivity of information that can be stored in the database, rather than reflecting the overall sensitivity of the information stored in the database. This is an important point, as it is different from what file labels mean, for example. The same argument holds true for the label on each table definition. Access to tables is granted only to users whose current session level dominates the sensitivity of the table definition. Therefore, any tuples inserted into the table must dominate the level of the table definition. The meaning of these labels and how they work is very similar to directory labels in multilevel file systems.

Normally, the maximum sensitivity of information that could be stored in the database is system-high. However, it is sometimes desirable to be able to explicitly define the maximum sensitivity of information that could be stored in a database or table. A trusted DBMS can support this capability by providing range labels for the database and/or each table, rather than just minimums. This would allow an explicit sensitivity range to be defined for the database and each table. This concept is similar to multilevel devices as described in the TCSEC. Minimum sensitivity labels are usually set equal to the session level of the user creating the database or table. The maximum can be set to any value which dominates the minimum, up to system-high.

Relationship Between Levels of Storage Objects

Based on the previous discussion it should be reasonably clear what relationships must exist between the sensitivity levels of the storage objects in the database. For clarity, these relationships are summarized here. This section describes current practice, but may not necessarily be true for all designs.

Database - The database label (or minimum label) will be dominated by the labels of all the table definitions, tuples and elements in the database. If the database has range labels, all table definitions, tuples and elements in the database will be within this range.

Table - The table definition label (or minimum label) will dominate the database label (or minimum label) and will be dominated by the maximum label if one exists. If the table has range labels, the range must reside within the range for the database.

Tuple - If tuples are labeled, they will dominate or lie within the range of the containing table definition's label(s).

Element -If elements are labeled, they will dominate or lie within the range of the containing table definition's label(s).

Trusted Processes

To avoid confusing the issue, the topic of trusted processes has been avoided during the previous discussion. However, it is important to note that the relationships previously described are not guaranteed to hold true if regrading or other trusted operations within the database are allowed. If something is regraded, inserted, or modified by a trusted process these relationships may not hold for the new or modified object. Tables, tuples or elements could be marked lower than the containing database or table. These items would then be readable, but could not be modified by untrusted processes because any modifications to them would be considered a write-down. If regrading causes these relationships to not hold true, it is not necessarily a violation of the security policy. For example, a trusted process could downgrade a tuple in a Secret table to Confidential. Once done, all untrusted processes must be at least Secret to gain access to the table, which prevents them from modifying the Confidential tuple because it would be considered a write down from Secret to Confidential. In general, these relationships should hold true for the majority of the information that is stored in the database. It is also possible for the security policy of the specific DBMS to require that these relationships hold in order to simplify the possible states of the database.

Label Constraints in Attribute Definitions

As stated before, a table definition defines the allowed contents of a table in terms of the number of attributes and each attribute's name and type. In a trusted DBMS, typically, additional attributes are automatically created by the DBMS based on the granularity of MAC provided and included within the table definition to hold the labels that mark the table's tuples or elements. As described previously, the labels of all tuples or elements within a table must dominate the sensitivity of the table definition. A DBMS could also provide the capability to define within a definition additional constraints on these label attributes. These constraints could be for specific values or a range of values, just like the values of normal attributes can be constrained within a table definition. For example, a table could be labeled Secret (which would set the minimum level for all tuples and elements in the table to Secret) while placing an additional constraint within the definition that required the label associated with all elements of the defined attribute 'Payload' to be at least Top Secret.

Views

Views are alternate ways of looking at the data stored in the database. A view definition is basically a query that has been given a name, which can then be used to refer to that particular view again in the future. Creation of a view definition or its execution does not alter the way the data is stored in the underlying data (base relations). Like all other objects, view definitions must

be labeled. These labels should dominate the labels of all tables accessed by the view in order to ensure that the view executes successfully. Otherwise the view may not be able to access the underlying base relations.

The label assigned to a view is up to the author of the view, based on the author's session level when the view is created. Views should be labeled based on the aggregate of the information that they access. If the author of a view believes that what is revealed by the view may be more sensitive than any single piece of information accessed by the view, then the view can be marked at a higher level indicative of the sensitivity of the aggregate of the data it accesses. This would restrict access to this view to only those users that are authorized to access the higher level data. However, if less cleared users can develop their own views, then this would not prevent a less cleared user from building an identical view that reveals the same information. Typically, however, databases are set up so only certain individuals can modify the basic structure of the database as well as develop new views. The classification of views is discussed in more detail in chapter 21 of [TLUN92].

View Based MAC

Since views are labeled, access to a view is restricted to those users that are cleared to the level of the view. This brings up the idea that MAC for a DBMS could be enforced based on the labels on views alone. The idea is that no labels would be provided on the base relations and all labeling would be provided by the labels on the view definitions [JWIL82, JGAR88]. There are a number of problems with this idea that have not been solved by research to date [Chapter 7 of TLUN92].

1) Assurance:Because of the complexity involved in providing high assurance for a DBMS based on view based MAC (since the view definition mechanism must be trusted, as well as the views developed by the individuals designing or maintaining the database), it would be very difficult to provide this capability with sufficient assurance. The main problem is that view definitions are typically written by database designers and administrators (DBMS users) rather than by DBMS product vendor personnel who go through a rigorous assurance process when they are developing a trusted DBMS for evaluation against the TDI.

2) Information Flow:More importantly, since views inherently overlap one another (i.e., can access data using more than one view), data can be implicitly labeled at different levels based on the labels for the views that can access data. These multiple labels for the same data create significant information flow problems which would clearly violate the MAC policy.

These as well as other reasons indicate why MAC is not done based on views alone. This is important to note since DAC frequently is provided based on views. The rationale for allowing this is described in the section on DAC.

Labeling Query Results

Queries (and views) allow users to extract information from the DBMS. Queries can only access information at a level that is dominated by the user's current session level. The results of these queries are returned to users as a set of data. In a trusted DBMS, these results are usually labeled. These labels indicate the sensitivity of the information returned to the user executing the query.

Correct generation of the labels for query results is not straightforward. Based on a strict interpretation of the MAC policy, these labels should be based on not only the label of the data returned by the query, but also the labels of all information used during the computation of the results. This information includes all data stored in the database that is accessed during the query as well as all queries/views used to compute the result. The resulting high water mark of all information accessed during the query should be returned as the minimum label of the query results in order to ensure that sensitive information is not inadvertently leaked. This is necessary because a user may be returned a result which is incorrectly marked at a level lower than the information that it actually reveals. The information should be marked at a higher level if the query outcome is affected by more sensitive intermediate information. The revelation of this incorrectly marked data results in a MAC violation because sensitive information is revealed to an insufficiently cleared individual.

If this strict interpretation of MAC is enforced by the DBMS, query results labels will tend to float upward as they are contaminated by 'touching' sensitive information during a query. The final results of a query may not depend on the contents of more sensitive information that was accessed during the query. Therefore, the results may not necessarily be more sensitive even though sensitive data was accessed. Much research has gone into how query results (as well as all data) should be labeled. Some thoughts on this area are presented in [Chapter 12 of TLUN92].

DAC in DBMSs

DAC for DBMSs is much more straightforward than MAC because of the existence of SQL and its requirements for DAC in databases. For DAC, DBMSs must enforce the same basic TCSEC requirement of "controlling access between named users and named objects". According to the TCSEC, the specific types of controls that must be provided are left up to the designer. However, because of the prevalence of SQL compliance, virtually all vendors of trusted DBMSs provide products which comply with SQL, in addition to meeting the requirements of the TCSEC. However, a vendor is not required to be SQL compliant to meet the requirements of the TCSEC. It should also be noted that there are ongoing SQL enhancements. However, this discussion on DAC presents the SQL requirements for DAC and describes what aspects of DAC are not specified by SQL and are left up to the DBMS vendor. This discussion on DAC is based on the description of security features in SQL in [TSQL89] and [TDAT89].

The following areas of DAC are specified by SQL:

- DAC on tables for individual users with the following privileges: SELECT, INSERT, DELETE, UPDATE, and REFERENCE, with or without the GRANT option which allows users to select whether a user possessing a privilege can pass this privilege on to another user.

- DAC on Views.

SQL does not specify:

- DAC for access to a database.

- The roles to be provided, the privileges to be assigned to these roles, or how they are to manage and how they are affected by the DAC mechanism.

Table Level DAC

In SQL, DAC is provided at the table level. Table level DAC in SQL is provided at the granularity of SELECT, INSERT, DELETE, UPDATE, and REFERENCE to individual users and to individual columns within the table for UPDATE and REFERENCE. Privileges can be granted by the owner of a table. For those that are interested, the semantics and syntax of these privileges are described in Appendix A of this module. In addition, these privileges can be granted with a 'WITH GRANT OPTION'. If a privilege is granted with this option, then the user receiving the privilege also gains the privilege to pass the privilege just granted on to other users. If this option is not set, then the user cannot pass the privilege on. Other privileges can be (and usually are) provided within a specific DBMS implementation.

Since table and view definitions are usually stored in tables as part of the database schema, the privileges mechanism for tables can be used to protect the database schema itself. Initially, when a database is created, only the owner of the database has any privileges to access or modify the database. All access rights to the database must be granted explicitly by the owner of the database, who is usually the database administrator. Therefore, the administrator can grant or not grant, the rights to create new tables or views, or modify existing ones, to other users.

Privileges are defined as part of the database schema using the GRANT clause and normally stored within a privilege table. How to revoke a privilege or set of privileges is currently not defined in SQL. Some implementations of SQL, such as IBM's DB2, include a REVOKE command. This revokes that user's access to the specified table or view. However, if the user had the 'WITH GRANT OPTION', any privileges that the user gave to other users are still intact. DBMS developers usually provide tools to help track down additional privileges that may exist. However, they are not required by the SQL standard.

View Based DAC

As stated previously, views provide an alternate way of looking at the data in the underlying base tables, or in other views. In SQL, the creator of a view must have access to all the tables or views accessed by the view when the view is created. Once created, access to a view may be granted by the owner to other users in the same manner that access to base tables can be granted. If a user has been granted access to a view, he can use the view to access the underlying base relations as defined by the view, even though the user may not have the privilege to access the base tables or views directly.

Views allow owners or administrators of data to grant users access to the information they control in very specific ways. Views can hide specific attributes, tuples or elements, or return statistical data based on the underlying data, thereby hiding the exact values that are stored. Views can also combine information from other tables to present information in different ways. The flexibility of views and their definitions allows DAC to be provided at a very fine level of granularity. To restrict the propagation of access rights through views, the creation of views is usually restricted to the owners or administrators of the data.

Database Level DAC

SQL does not specify how access to the entire database is to be controlled. The details are left up to the implementor of the DBMS. Potential access types that could be provided include 'Full Access', 'No Access', 'Read Only', and 'Update Only'. Initial access to the database can also be used to assign a specific role to an individual based on the account they use to access the database. These details are all left up to the DBMS developer.

Roles

Another area that SQL leaves up to the implementor of the DBMS is what roles are to be provided and how these roles affect the management and enforcement of the DAC policy. Roles such as Database Administrator and Database Security Officer are typically provided. These roles are usually granted certain privileges which allow them to manage and/or override the DAC mechanism during the execution of their duties. The exact details of these capabilities are left up to the DBMS developer. The TCSEC and TDI require specific roles to be provided by the DBMS at level B2 and above. Thus, compliance with SQL is not necessarily sufficient for a DBMS to meet the TCSEC requirements for DAC at these higher assurance levels.

Identification and Authentication in DBMSs

As stated in the TDI, the I&A requirements for a trusted DBMS are identical to that required for a trusted operating system (OS). The basic requirement is to identify and authenticate individual users when they first request access to the DBMS and then to use the authenticated identity as the basis for all access control decisions. This process is basically the same for an OS as it is for a DBMS. What makes it somewhat different is not in the policy area but in the implementation of the policy because of the fact that the DBMS usually runs

on top of an underlying OS. If a user has been authenticated to the OS, then this authentication data, the user's current session level, and any user privileges could be shared between the OS and the DBMS. However, sharing is not required by the TDI. The TDI allows the OS and DBMS TCBs to maintain separate I&A databases and mechanisms. These implementation details are discussed more in TDI Module Three.

Audit in DBMSs

The TDI provides additional detail on the audit requirements for a Trusted DBMS on pp. 44-46. It states that the DBMS must be able to audit "all mediated accesses which are visible to the user. That is, each DAC policy decision and each MAC policy decision shall be auditable." Operations by trusted software which are totally transparent to the user need not be auditable. This is the same as the TCSEC requirement. However, because of the granularity and number of 'objects' that are being protected by the DBMS TCB (tables, tuples, elements) and the richness of the discretionary access controls (as described above), a DBMS must audit most database operations that a user can perform while using a DBMS.

These requirements require large amounts of audit data to be generated and stored by the DBMS TCB. The TDI allows security audit data to be kept separate from the security audit data generated by an underlying OS. It also allows the DBMS to generate multiple audit logs within its own domain. It must be possible to correlate all DBMS and OS logs, which can be done in real-time or after the fact. The TDI also allows for a DBMS to provide "a selectable capability to reduce the number of audit records generated in response to queries that involve many access control decisions". This allows database administrators to reduce the amount of audit data generated to an acceptable level.

Integrity Policies

Integrity policies are very important to trusted DBMSs even though they are not required by the TDI/TCSEC and are not heavily scrutinized during a TDI evaluation by NSA. A concise discussion of this topic is difficult because of the many different meanings people have/envision when the term 'integrity' is used. Integrity can be defined as ensuring that the database is internally consistent (with some a priori set of criteria) and correctly reflects the real world (again, based on some set of criteria). This usage of integrity is also referred to as data integrity.

The following are examples of data integrity 'policies' that are typically provided by a DBMS:

- entity integrity
- referential integrity
- integrity constraints (data type, set of values, compatibility, etc.)

In SQL, entity integrity is the requirement that all primary keys of a table are non-null and unique. Whenever a new tuple is inserted, or a primary key is updated, a check is made that ensures that the key value is not duplicated. In SQL, referential integrity requires that all foreign keys reference an existing primary key. A foreign key, is an element that references (matches) the primary key of another tuple. A referential integrity constraint is automatically invoked whenever a foreign key is created. Integrity constraints can also be explicitly specified for each attribute in their table definition. These constraints can specify data type, specific values or range of values, uniqueness properties, and other constraints. These policies and others are discussed in [JWIS90].

Another integrity policy that could be enforced is the Biba "strict integrity" policy as described in [JBIB77]. This is a label-based (mandatory) approach that is sometimes provided by trusted operating systems. If provided by the underlying OS, the DBMS may also need or desire to support this policy as well. The basic premise of this policy is to keep low integrity subjects and their data from contaminating higher integrity data. This mechanism can be provided by using integrity levels and categories in a manner similar to mandatory security levels and categories.

Another example of integrity is described in the Clark-Wilson model using "triples" as an enforcement mechanism and the concept of "separation of duties" which involves the concept of roles. In this model, each user is assigned specific "transactions" that the user can execute on specific sets of data based on the user's role or job function. To provide separation of duties, the administrator of the system must ensure that different individuals are assigned the rights to execute different transactions for the same set of data. Thus, for a given operation, which takes multiple transactions to complete, different individuals would have to be involved to complete the entire operation. This separation of duties is intended to reduce fraud and abuse. This type of policy fits naturally into a DBMS where DBMS operations (transactions) are to be separated and controlled.

Many other integrity policies are possible. However, it is beyond the scope of this course to go into more detail in this area. The important thing to note here is that, because of the nature of database applications, integrity requirements typically arise that must be supported by the DBMS. These integrity requirements, which are not required by the TCSEC, may have serious security ramifications as described in the following section.

Integrity vs. Security

Providing integrity and/or security are important aspects of database management. However, because of the nature of DBMS integrity and security requirements, similarities as well as conflicts arise when both policies are simultaneously enforced.

Security policies (both MAC and DAC) which can prevent access or restrict access to read-only, inherently provide a certain level of integrity because modify access can be restricted. Thus some integrity is provided by preventing

unauthorized modifications. However, restricting who can perform modifications is not always sufficient to ensure that certain integrity requirements are met. Therefore, integrity policies are developed which levy additional requirements such as those described previously.

One of the main problems when trying to enforce MAC on a DBMS is that entity integrity enforcement is difficult without creating covert signaling channels. If a lower level tuple is created with the same primary key as a higher level tuple, the system must prevent this in order to preserve the uniqueness of the keys (entity integrity). This however reveals the existence of the higher level tuple, which is a covert channel. Trusted DBMS researchers solved this problem with the concept of polyinstantiation. Primary keys are expanded to include the tuple label in addition to the original primary key. Therefore the two tuples can now be distinguished by their sensitivity labels. In this manner multiple tuples with the same original primary key but different sensitivity labels can exist simultaneously, hence the term polyinstantiation. Polyinstantiation solves the entity integrity problem, but creates other integrity problems because multiple tuples describing the same entity can now exist and it may not be clear which tuples accurately reflect the real world. This is currently a research topic and is discussed in more detail in TDI Module 4.

Integrity constraints on attributes that cross access classes can also create covert channels. For example, if a table describing cargo planes places a limit of 500 lbs on its cargo and it is carrying both Unclassified and Secret cargo, an uncleared user trying to add additional cargo may learn that more sensitive cargo is on board if the system prevents him from loading up to 500 lbs of Unclassified cargo. If this constraint is enforced across access classes then less cleared users will be able to determine the exact weight of the sensitive cargo based on how much Unclassified cargo can be stored on the plane. This type of problem is discussed in more detail in the inference section of TDI Module 4.

These and other issues regarding integrity vs. security need to be addressed by the developer of the DBMS. The tradeoffs between providing flexibility to the users, e.g., can polyinstantiate, can define integrity constraints across access class boundaries, need to be balanced against the security requirements. These areas may not be solvable by the DBMS developer for all potential applications. DBMSs can provide the capability for these mechanisms to their users and then leave it up to the users to decide what they want. This removes the burden of decision from the DBMS developer and provides more flexibility than a fixed solution. Methods for resolving some security vs. integrity conflicts are discussed in [JMAI91].

Concurrency and Recovery

The ability to support concurrent users and to recover from aborted or interrupted transactions are standard capabilities for DBMSs today. These capabilities must be carefully considered when being supported by a trusted DBMS. Concurrency controls (i.e., locking) can cause significant covert

channels to be introduced within the database. Recovery from aborted transactions can also introduce covert channels.

Concurrency controls inherently cause users to affect one another by delaying or blocking completion of transactions until other transactions are completed. If these delays can be detected by a user (i.e., because of time or because rollbacks are initiated) then some information about one user can be transmitted to another. In a multilevel system, this information flow could be in violation of the MAC policy, thus creating a covert channel. Some research has been done in providing covert channel free locking mechanisms. One such approach is described in [JMCD92].

Transactions can abort for any number of reasons, only some of which indicate that there was an actual failure in the DBMS. Most transactions abort when the transaction detects a condition which the author of the transaction believes will interfere with correct completion of the transaction. System crashes can also interfere with the completion of transactions. It is the DBMS's responsibility to restore the database to the same state it was in before the transaction began. The multilevel security requirement makes it more difficult to correctly accomplish this task without introducing covert channels during the recovery process. One approach to secure recovery controls is presented in [JKAN92].

Required Readings

The required readings are supplied as part of the source material for the module. These readings, and the module overview, provide all the material covered by the module test questions.

DTDI91 National Computer Security Center, *Trusted DBMS Interpretation of the Trusted Computer System Evaluation Criteria* (TDI), NCSC-TG-021, Version-1, April 1991.

Sections TC-5.2.1, IR-2.1, & IR-3 describes security policy requirements as interpreted for databases. Appendix A presents a summary of interpretations by class, including the security policy requirements.

TLUN92 Research Directions in Database Security, Teresa Lunt, Editor, Springer Verlag, 1992. [Chapters 19 and 21]

Chapter 19 discusses security policies in trusted DBMS. Sections 19.2.4, 19.3, and 19.4 discuss specific problems that typically are present in DBMS security policies. Chapter 21 talks about the classification of metadata and views.

Other Related Readings

TLUN92 Research Directions in Database Security, Teresa Lunt, Editor, Springer Verlag, 1992. [Chapters 7, 11, and 12.]

JMAI91 B. Maimone, and R. Allen, "*Methods for Resolving the Security vs. Integrity Conflict*", Proceedings of the Fourth RADC Database Security Workshop, April 1991.

JWIS90 Simon Wiseman, *"The Control of Integrity in Databases"*, Fourth IFIP WG 11.3 Workshop on Database Security, Sept. 1990.

JBIB77 K. J. Biba, *"Integrity Considerations for Computer Systems"*, Mitre TR-3153, Mitre Corp., Bedford, MA, April 1977.

JCLA87 D. D. Clark and D. R. Wilson, *"A Comparison of Commercial and Military Computer Security Practices"*, 1987 IEEE Symposium on Security and Privacy, 1987.

JCOD70 Codd, E. F., *"A Relational Model of Data for Large Shared Data Banks"*, Communications of the ACM, June 1970.

JDEN88 D. Denning, et al, *"The SeaView Security Model"*, 1988 IEEE Symposium on Security and Privacy, 1988.

JGAR88 C. Garvey and A. Wu, *"ASD_Views"*, 1988 IEEE Symposium on Security and Privacy, 1988.

JKAN92 I. Kang and T. Keefe, *"Recovery Management for Multilevel Secure Database Systems"*, Proceedings of Sixth IFIP WG 11.3 Working Conference on Database Security, August 1992.

JLUN88 T. F. Lunt, et al, *"Final Report Vol. 1: Security Policy and Policy Interpretation for a Class A1 Multilevel Secure Relational Database System"*, Computer Science Laboratory, SRI International, Menlo Park, California, 1988.

JMCD92 J. McDermott and S. Jajodia, *"Orange Locking, Channel-Free Database Concurrency Control Via Locking"*, Proceedings of Sixth IFIP WG 11.3 Working Conference on Database Security, August 1992.

JWIL82 J. Wilson, *"Views as the Security Objects in a Multilevel Secure Relational Database Management System"*, 1988 IEEE Symposium on Security and Privacy, 1988.

DREC91 National Computer Security Center, *A Guide to Understanding Trusted Recovery in Trusted Systems*, NCSC-TG-022, Version 1, 30 Dec 1991.

TDAT89 C. J. Date, A Guide to the SQL Standard, 2nd Edition, Addison Wesley, 1989.

TSQL89 Database Language - SQL with Integrity Enhancement, ANSI X3.135-1989, American National Standards Institute, NY, NY, 1989.

Appendix A: DBMS DAC as specified by SQL

4.15 Privileges[2]

A privilege authorizes a given category of <action> to be performed on a specified table or view by a specified <authorization identifier>. The <action>s that can be specified are INSERT, DELETE, SELECT, UPDATE, and REFERENCES.

An <authorization identifier> is specified for each <schema> and <module>.

The <authorization identifier> specified for a <schema> shall be different from the <authorization identifier> of any other <schema> in the same environment. The <authorization identifier> of a <schema> is the "owner" of all tables and views defined in that <schema>.

Tables and views are designated by <table name>s. A <table name> consists of an <authorization identifier> and an <identifier>. The <authorization identifier> identifies the <schema> in which the table or view designated by the <table name> was defined. Tables and views defined in different <schema>s can have the same <identifier>.

If a reference to a <table name> does not explicitly contain an <authorization identifier>, then the <authorization identifier> of the containing <schema> or <module> is specified by default.

The <authorization identifier> of a <schema> has all privileges on the tables and views defined in that <schema>.

A <schema> with a given <authorization identifier> may contain <privilege definition>s that grant privileges to other <authorization identifier>s. The granted privileges may apply to tables and views defined in the current <schema>, or they may be privileges that were granted to the given <authorization identifier> by other <schema>s. The WITH GRANT OPTION clause of a <privilege definition> specifies whether the recipient of a privilege may grant it to others.

A <module> specifies an <authorization identifier>, the <module authorization identifier>, which shall have the privileges specified for each <SQL statement> in the <module>.

---

2. From FIPS Pub 127-1 p. 14, which is a copy of ANSI Standard X3.135-1989.

Format[3]

<privilege definition> ::=
GRANT <privileges> ON <table name>
TO <grantee> [{, <grantee> }...]
[WITH GRANT OPTION]

<privileges> ::=
ALL PRIVILEGES
| <action> [{, <action> }...]

<action> ::=
SELECT | INSERT | DELETE
| UPDATE [(<grant column list>)]
| REFERENCES [(<grant column list>)]

<grant column list> ::=
<column name> [{, <column name> }...]

<grantee> ::=
PUBLIC | <authorization identifier>

---

3. This BNF notation for the syntax of a GRANT is from FIPS Pub 127-1, p. 68.